Epiphenomenal Grammar Acquisition with GSG

Marsal Gavaldà
Interactive Systems, Inc.
1900 Murray Ave. Suite 203
Pittsburgh, PA 15217, U.S.A.
marsal@interactivesys.com

Abstract

As a step toward conversational systems that allow for a more natural human-computer interaction, we report on GSG, a system that, while providing a natural-language interface to a variety of applications, engages in clarification dialogues with the end user through which new semantic mappings are dynamically acquired. GSG exploits task- and language-dependent information but is fully task- and language-independent in its architecture and strategies.

1 Introduction

As conversational systems move from the realm of science fiction and research labs into people's everyday life, and as they evolve from the plain, system-directed interactions à la *press or say one* of so-called interactive voice response systems based on isolated-word recognizers and fixed-menu navigation, to the more open, mixed-initiative dialogues carried out in spoken dialogue systems based on large-vocabulary continuous speech recognizers and flexible dialogue managers (see, e.g., (Allen et al., 1996; Denecke, 1997; Walker et al., 1998; Rudnicky et al., 1999; Zue et al., 2000)), the overall experiential quality of the human-computer interaction becomes increasingly important. That is, beyond the obvious factors of speech recognition accuracy and speech synthesis naturalness, the most critical challenge is that of providing conversational interactions that feel natural to human users (cf. (Glass, 1999)). This, we believe, mainly translates into building systems that possess some degree of linguistic, reasoning, and learning abilities.

In this paper we report on GSG, a conversational system that partially addresses these issues by being able to dynamically extend its linguistic knowledge through simple, natural-language only interactions with non-expert users: On a purely on-need basis, i.e., when the system does not understand what the user means, GSG makes educated guesses, poses confirmation and clarification questions, and learns new semantic mappings from the answers given by the users, as well as from other linguistic information that they may volunteer. GSG provides, therefore, an extremely robust interface, and, at the same time, significantly reduces grammar development time because the original grammar, while complete with respect to the semantic representation of the domain at hand, need only cover a small portion of the surface variability, since it will be automatically extended as an epiphenomenon of engaging in clarification dialogues with end users.

2 Brief System Description

As sketched in Figure 1, GSG is a conversational[1] system built around the SOUP parser (Gavaldà, 2000).

GSG's principal (and possibly sole) knowledge source is a task-dependent, semantic context-free grammar (the Kernel Grammar). At run-time, the Grammar is initialized as the union of the Kernel Grammar and, possibly, the User Grammar Δ (user-dependent rules learned in previous sessions). The Grammar gives rise to the Ontology and to a parsebank (collection of parse trees), which, together with a possible Kernel Parsebank, becomes the Parsebank, from which the statistical Prediction Models are trained. The Ontology is a directed acyclic graph automatically derived from the Grammar in which the nodes correspond to grammar nonterminals (NTs) and the arcs record immediate dominance relation, i.e., the presence of, say, NT_i in a right-hand side (RHS) alternative of NT_j will result in an arc from NT_i to NT_j. Nodes are annotated as being "Principal" vs. "Auxiliary" (via naming convention), "Top-level" vs. "Non-top level" (i.e., whether they are starting symbols of the grammar), and with having "Only NT daughters" vs. "Only T daughters" vs. "Mixed"; arcs are annotated as being "Is-a" (estimated from being the only non-optional NT in a RHS alternative) vs. "Expresses" links, "Always-required" vs. "Always-optional" vs. "Mixed," and "Never-repeatable" vs.

[1] In the work reported here, GSG's interactions are text-based (keyboard as input, text window as output), but GSG is being integrated with both a speech recognizer and a speech synthesizer.

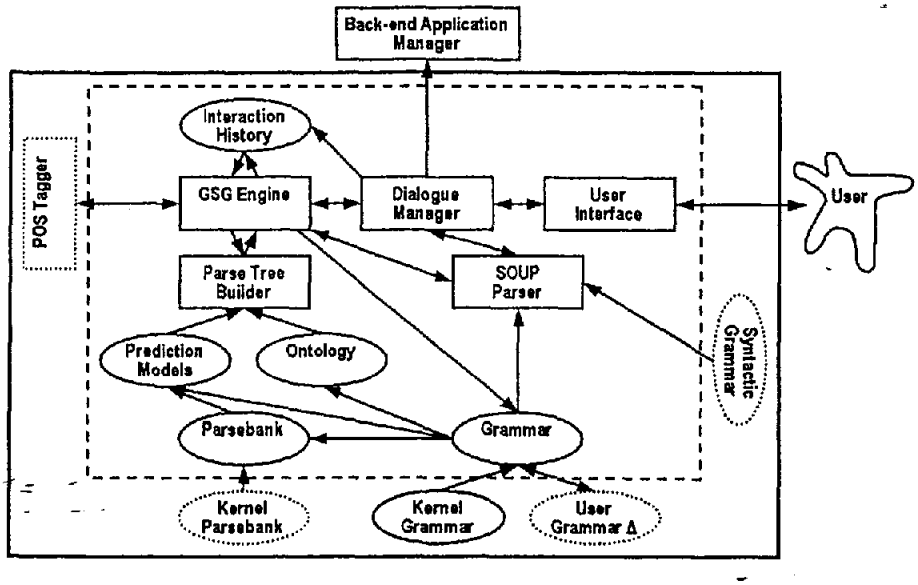

Figure 1: GSG's system diagram. Ovals enclose knowledge sources, rectangles modules, and arrows indicate information flow. Dashed components are optional.

Strategy	Knowledge Source
All-top Parsing	Grammar
Anchor Mother Predictions	Prediction Models
Required/Is-a/... Daughters Search	Ontology
Verbal Head Search	POS Tagger, Ontology
Parser Predictions	Grammar

Table 1: List of GSG's main prediction and learning strategies.

"Always-repeatable" vs. "Mixed". Also, a topological sort[2] on the nodes is computed to derive a general-to-specific partial order of the NTs.

A full system description is beyond the scope of this paper, but, very briefly, the User Interface mediates all interactions with the end-user, the stack-based Dialogue Manager keeps track of current and past utterances and ensuing clarification dialogues, and, together with the History Interaction, ensures that no answered question is asked again. The GSG Engine manages the core of the systems' "intelligence," namely hypothesizing interpretations (together with the Parse Tree Builder) and on-line learning of semantic mappings.

3 Example Dialogue

To illustrate the workings of GSG, let's analyze an example interaction in an e-mail client task. Figure 2 shows the example dialogue, Figure 3 presents a relevant fragment of the semantic context-free grammar[3] used to analyze the input, and Table 1 above lists the main prediction and learning strategies employed.

The user inputs (U1) and, as it turns out, neither *retrieve*, nor *bob*, nor *christmas* are in the grammar and no full parse of (U1) can be found. Rather than responding with a *I don't understand — please rephrase*, GSG sets its predictive machinery in motion with the goal of constructing a full interpretation for (U1).[4]

The first step is to reparse (U1) in a mode in which all NTs are considered top-level, i.e., able to stand at the root of a parse tree. This produces a set of still unparsed words, such as *retrieve*, and parse subtrees, such as *from bob* being parsed under SENDER (via grammar rules (R6) to (R10)) and, ambiguously, under `[sourceFolder]` as well (via grammar rules (R11) and (R12)). All of it (i.e., the full content of the chart) is taken as evidence by the Prediction Models to postulate the overall meaning of the original utterance. In this case (see (U2)

[2]Requires that the grammar be acyclic.

[3]Containing, approximately, 300 NTs, 500 Ts, and 800 RHS alternatives, out of which about 55% is dedicated to cover temporal and numeric expressions, and digits and letters.

[4]The hypothesized interpretation assumes that a single parse tree will cover the entire problematic sentence.

(U1) U: retrieve all messages from bob that were sent after christmas
(U2) S: I don't understand right away what you mean but let me guess...
"retrieve all messages from bob that were sent after christmas" is a way to express:
1. send mail
2. reply mail
3. read mail
0. None of the above
(U3) U: none
(U4) S: "retrieve all messages from bob that were sent after christmas" is a way to express:
1. move mail
2. list mail
0. None of the above
(U5) U: list
(U6) S: By "from bob" do you mean:
1. source folder
2. sender
(U7) U: sender
(U8) S "christmas" is a way to express:
1. yesterday (date relative yesterday)
2. tomorrow (date relative tomorrow)
3. today (date relative today)
0. None of the above
(U9) U: by christmas i'm referring to december twenty fifth
(U10) S: Thanks for teaching me the meaning of "christmas"!
Thanks for teaching me the meaning of "retrieve all messages from bob that were sent after christmas"!
I understand "retrieve all messages from bob that were sent after december twenty fifth"
(U11) U: retrieve last email to mary before christmas
(U12) S: I understand "retrieve last email to mary before december twenty fifth"

Figure 2: Example dialogue between a user (U) and the system (S) on an e-mail client task.

to (U5)[5]) the suggestions of the Prediction Models are not particularly accurate (the correct choice is presented only in fifth place), but, considering that the head verb (*retrieve*) is not even in the grammar, such a response to (U1) is definitely better than giving up. The effect of (U5) is to select `[listMail]` as (U1)'s "anchor mother" (logical root of the overall interpretation). But to complete the parse tree a few details still need to be filled in. To that effect (U6) is generated to disambiguate *from bob* and (U8) to find the right mapping for *christmas*. The reasoning behind the rather puzzling choices offered by (U8) comes from applying the Parser Predictions strategy: given the context in which an unparsed sequence (in this case, single word) *christmas* appears, i.e., the subtree `DATE_AFTER_PRE` covering *after* (via (R14)), the grammar is traversed to find likely continuations of the context (left context only in this case). Since `DATE_AFTER_PRE` can be immediately followed by `[dateAfter]` (see (R13)) that makes `[dateAfter]` a candidate to cover the unparsed sequence *christmas*. However, since, according to the Ontology, `[dateAfter]` does not allow terminals as immediate daughters, a search is performed to find NTs under `[dateAfter]` that permit it. In this case (via (R15) to (R19)) it suggests *yesterday*, *tomorrow*, etc.[6] The user, though, realizing that the system does not directly understand *christmas*, volunteers (U9)[7], from which the mapping (M2) in Figure 4 is learned.

At this point one may wonder about the fate of the unparsed word *retrieve*, since no question was asked about it. The answer is that GSG need not ask about every single prediction, if the confidence value is high enough. In this case, as soon as `[listMail]` was established (in (U5)) as the anchor mother, a Verbal Head Search strategy was launched to see whether, among the unparsed words, a verb was found that could be placed in a mostly-verb NT[8] directly under

[5]The options presented in (U2) and (U4) are generated at the same time, the only reason why they are split is to prevent overwhelming the end user, who may be hearing the choices spoken over the telephone. Also, note that in (U3) the user could have also said *zero*, or *none of the above* and achieve the same result — or, alternatively, they could have volunteered information as in (U9).

[6]In fact it suggests `[DATE_RELATIVE:yesterday]`, `[DATE_RELATIVE:tomorrow]`, etc, but it presents an example automatically generated from such NTs.

[7]Obviously "the meaning of Christmas" (cf. cheerful (U10)) may be much more profound than a shorthand for December 25 — but, alas, conveying that is well beyond the simple grammar presented here.

[8]A "verbness" ratio is automatically computed for each candidate NT. by running the POS tagger on automatically generated sentences from the NTs in question. (We used a

(R1)	[listMail]	⟵	*VERB_DESIRE LIST *TO_FOR_ME +MAIL_ARGUMENTS
(R2)	[moveMail]	⟵	*VERB_DESIRE MOVE *+MAIL_ARGUMENTS *[sourceFolder] [destinationFolder]
(R3)	LIST	⟵	*list* | *get*
(R4)	MOVE	⟵	*move*
(R5)	MAIL_ARGUMENTS	⟵	SENDER | RECIPIENT | SUBJECT | DATE | MESSAGE_IDX | ...
(R6)	SENDER	⟵	*SENDER_PRE [sender]
(R7)	SENDER_PRE	⟵	*from* | *by*
(R8)	[sender]	⟵	[name:STRING] | [emailAddress:STRING]
(R9)	[name:STRING]	⟵	PERSON_OR_INSTITUTION_NAME | MAILING_LIST_NAME
(R10)	PERSON_OR_INSTITUTION_NAME	⟵	+WILDCARD
(R11)	[sourceFolder]	⟵	*from* [folderName:STRING] *FOLDER
(R12)	[folderName:STRING]	⟵	WILDCARD
(R13)	[dateRange]	⟵	(DATE_AFTER_PRE [dateAfter]) | (DATE_BEFORE_PRE [dateBefore]) | ...
(R14)	DATE_AFTER_PRE	⟵	*after* | *from* | *since*
(R15)	[dateAfter]	⟵	[datePoint:DATE]
(R16)	[datePoint:DATE]	⟵	+DATE_POINT_ARGUMENT
(R17)	DATE_POINT_ARGUMENT	⟵	[datePoint:DATE_RELATIVE] | [datePoint:DATE_FIXED] | ...
(R18)	[datePoint:DATE_RELATIVE]	⟵	[DATE_RELATIVE:yesterday] | [DATE_RELATIVE:tomorrow] | ...
(R19)	[DATE_RELATIVE:yesterday]	⟵	*yesterday*

Figure 3: Grammar fragment for an e-mail client task. '*' indicates optionality of adjacent token, '+' repeatability, and '|' separates RHS alternatives. Terminals are *italicized*. WILDCARD is a special NT that matches any out-of-vocabulary word or any in-vocabulary word present in a list for that purpose.

[listMail]. The result was highly positive and led to the acquisition of the RHS alternative (M1).

It is worth mentioning here that there are two kinds of mappings that GSG learns: RHS alternatives and subtree mappings. Learning new RHS alternatives is the preferred way because the knowledge can be incorporated into the Parsebank (and, in turn, into the Prediction Models). That is the effect of adding (M1) to the Grammar: Since the Parsebank and the Prediction Models are updated on-line, the presence of the word *retrieve* in subsequent utterances becomes a strong indicator of LIST and, associatively, of [listMail]. However, when the source expression can not be mapped into the desired target structure via grammar rules, as in (M2), the only solution is to remember the equivalence. This kind of learning, although definitely useful since the meaning of the source expression will be henceforth remembered, cannot be incorporated into the Prediction Models.

Right after (U9), (U1) is considered fully understood and the interpretation is automatically mapped into the feature structure (FS1)[9] in Figure 5, which is then shipped to the Back-end Application Manager.

Finally, when (U11) comes in, a correct analysis is produced thanks to the mappings just learned from (U1),[10] and (FS2) in is generated.

4 Discussion

The example above illustrates the philosophy of GSG,[11] namely, to exploit task and linguistic knowledge to pose clarification questions in the face of incomplete analyses,[12] build correct interpretations, and acquire new semantic mappings. Thus, a contribution of GSG, is the demonstration that from a simple context-free grammar, with a very lightweight formalism, one can extract enough information (Ontology, Parsebank, Parser Predictions strategy) to conduct meaningful clarification dialogues. Note, moreover, that such dialogues occur entirely *within* GSG, with the Back-end Application Manager receiving only finalized feature structures.[13]

Another advantage is the ease with which natural-language interfaces can be constructed for new domains: Since all the task and linguistic knowledge is extracted from the grammar,[14] one need only develop a Kernel Grammar that models the domain at

modified version of Brill's tagger (Brill 1994).)

[9]The mapping is simply a removal of auxiliary NTs from the parse tree, plus value extraction of dates, numbers and strings from certain subtrees, e.g., subtree in (M2) becomes the substructure under datePoint in (FS1).

[10]Note that rule [listMail] ⟵ LIST +MAIL_ARGUMENTS (extracted from the final interpretation of (U1)) would have been learned too, but its subsumption by existing rule (R1) was automatically detected.

[11]Based on the pioneering work of (Lehman, 1989).

[12]Detected by a lack of interpretation, excessively fragmented interpretation, or by being told by the end user that the automatically generated paraphrase of their input is not what they meant.

[13]Of course, prediction accuracy can improve if the Back-end Application Manager can be incorporated as a knowledge source to, for example, contribute in the ranking of hypotheses, but the point is that it is not necessary and that, as long as the capabilities of the back-end application are adequately modeled by the Grammar, the construction of the correct interpretation can be performed within GSG alone.

[14]Except for the POS Tagger and the Syntactic Grammar.

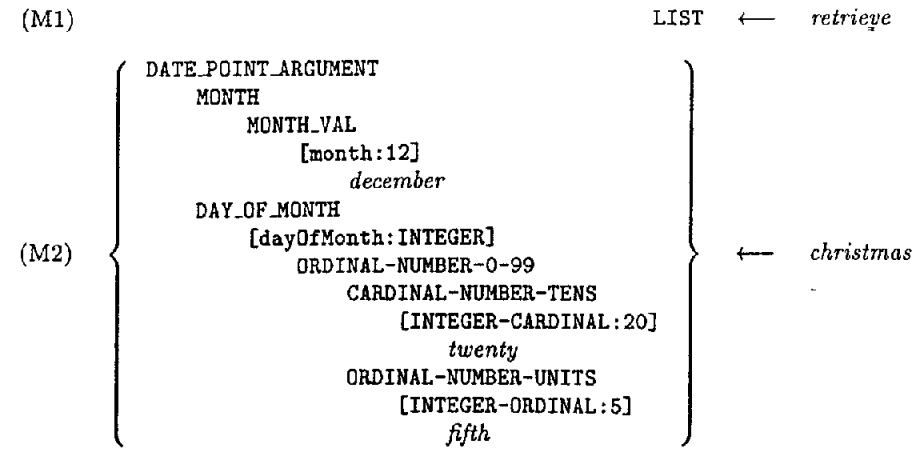

Figure 4: Mappings learned from the dialogue in Figure 2.

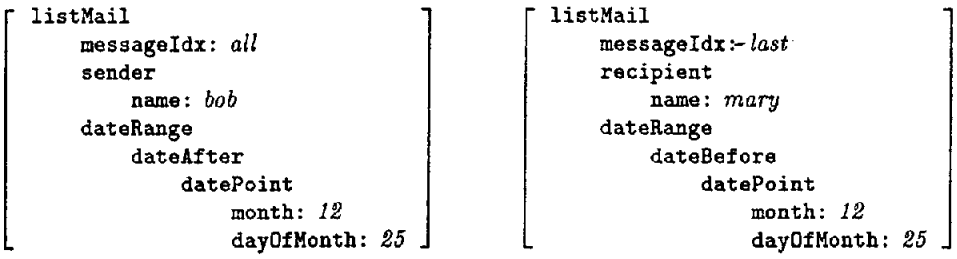

Figure 5: Feature structures sent to the Back-end Application Manager after (U10) and (U12) in Figure 2.

hand via its NTs[15] but need not provide a high coverage of the utterances possible in the domain (data which may not be available anyway). Also, reuse of existing grammar modules for, e.g., dates and numbers, is straightforward.

However, a fear of letting the end user (indirectly) modify a grammar is that the grammar may grow untamed and become filled with new rules that disrupt the Kernel Grammar. To prevent that, besides the careful construction of interpretations via the strategies described above, GSG employs two safety mechanisms: before a rule is added to the grammar, it is checked whether it introduces ambiguity to the grammar,[16] and whether it disrupts existing (correct) interpretations.[17] In this way, some of the new rules may have to be discarded, but at least the health of the grammar is preserved.[18]

Another concern may be that the new mappings end up generating feature structures that are not understood by the Back-end Application Manager. To avoid that, GSG only allows a principal NT to be dominated by another principal NT if such dominance relation is licensed by the Kernel Grammar. This guarantees that all resulting feature structures be structurally correct (although they may contain unexpected atomic values).

A current limitation of GSG lies in the difficulty of segmenting long sequences of unparsed words: GSG uses POS tagging followed by noun-phrase bracketing (via parsing with a shallow Syntactic Grammar), which represents an improvement over the Single Segment Assumption (cf. (Lehman, 1989)), but is still far from perfect and can disrupt the ensuing clarification dialogue. Also, the number of questions that the system can pose as it builds an interpre-

[15] Knowledge of, e.g., how the Ontology is computed helps, but it coincides with the most natural way of writing well-structured, context-free semantic grammars.

[16] Accomplished by using the SOUP parser in yet another mode: parsing of RHSs (expanded to RHS paths) instead of terminals. In this case, existence of a parse tree covering an entire RHS path indicates ambiguity. Note that if all RHS paths of the new rule can be parsed under the current RHS of the new rule's left-hand side, then the new rule is *subsumed* by the existing RHS and can therefore be discarded (cf. note 10).

[17] Achieved by reparsing (a subset of) the Parsebank. Note that SOUP can typically parse in the order of 100 utterances per second (cf. (Gavaldà, 2000)).

[18] Assuming minimally coöperative and consistent users.

tation, may, in occasion, exceed the patience of the end user (but the command *cancel* is always understood).

The hardest problem we have encountered so far is typical of natural-language interfaces but is exacerbated in GSG (as it treats every unparsable sentence as an opportunity to learn), and that is the difficulty of identifiying *in-domain* end-user sentences that go beyond the capabilities of the end application, or, in other words, are not expressible in the grammar.

Finally, as GSG becomes fully integrated with a speech recognizer, it remains to be seen how an optimal point in the tradeoff between the wide coverage but relatively low word recognition accuracy obtained with a loose dictation grammar, and the narrow coverage but high word accuracy achieved with a tight, task-dependent grammar, can be found, and how the degradation of the input is going to affect GSG's behavior.

Overall, however, we believe that GSG, by virtue of its built-in robustness, minimal initial knowledge requirements, and learning abilities, begins to embody the kind of qualities that are necessary for conversational systems, if they are to provide, without exorbitant development effort, an interaction thay feels truly natural to humans.

References

Allen, James, et al. (1996). Robust Understanding in a Dialogue System. In *Proceedings of ACL-1996.*

Brill, Eric. (1994). Some Advances in Part of Speech Tagging. In *Proceedings of AAAI-1994.*

Denecke, Matthias. (1997). An Information-based Approach for Guiding Multi-modal Human-Computer Interaction. In *Proceedings of IJCAI-1997.*

Gavaldà, Marsal. (2000). SOUP: A Parser for Real-world Spontaneous Speech. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT-2000).*

Glass, James. (1999). Challenges for Spoken Dialogue Systems. In *Proceedings of the 1999 IEEE ASRU Workshop.*

Lehman, Jill. (1989). *Adaptive Parsing: Self-extending Natural Language Interfaces.* Ph.D. dissertation, School of Computer Science, Carnegie Mellon University.

Rudnicky, Alex, et al. (1999). Creating Natural Dialogs in the Carnegie Mellon COMMUNICATOR System. In *Proceedings of Eurospeech-1999.*

Walker, Marilyn, et al. (1998). Learning Optimal Dialogue Strategies: A Case Study of a Spoken Dialogue Agent for Email. In *Proceedings of COLING/ACL-1998.*

Zue, Victor, et al. (2000). JUPITER: A Telephone-Based Conversational Interface for Weather Information. In *IEEE Transactions on Speech and Audio Processing*, Vol. 8 , No. 1.